Introduction to Database Systems
CSE 444

Lecture 2: SQL

Announcements

- Project 1 & Hw 1 are posted on class website
  - Project 1 due in two weeks (11:45pm)
  - Homework 1 due in three weeks (in class)
  - Remember: time goes by very fast! Start early!
- Your database accounts will be ready today
- Nodira's office hours are posted on website
- Chairs should no longer be squeaking!

Outline

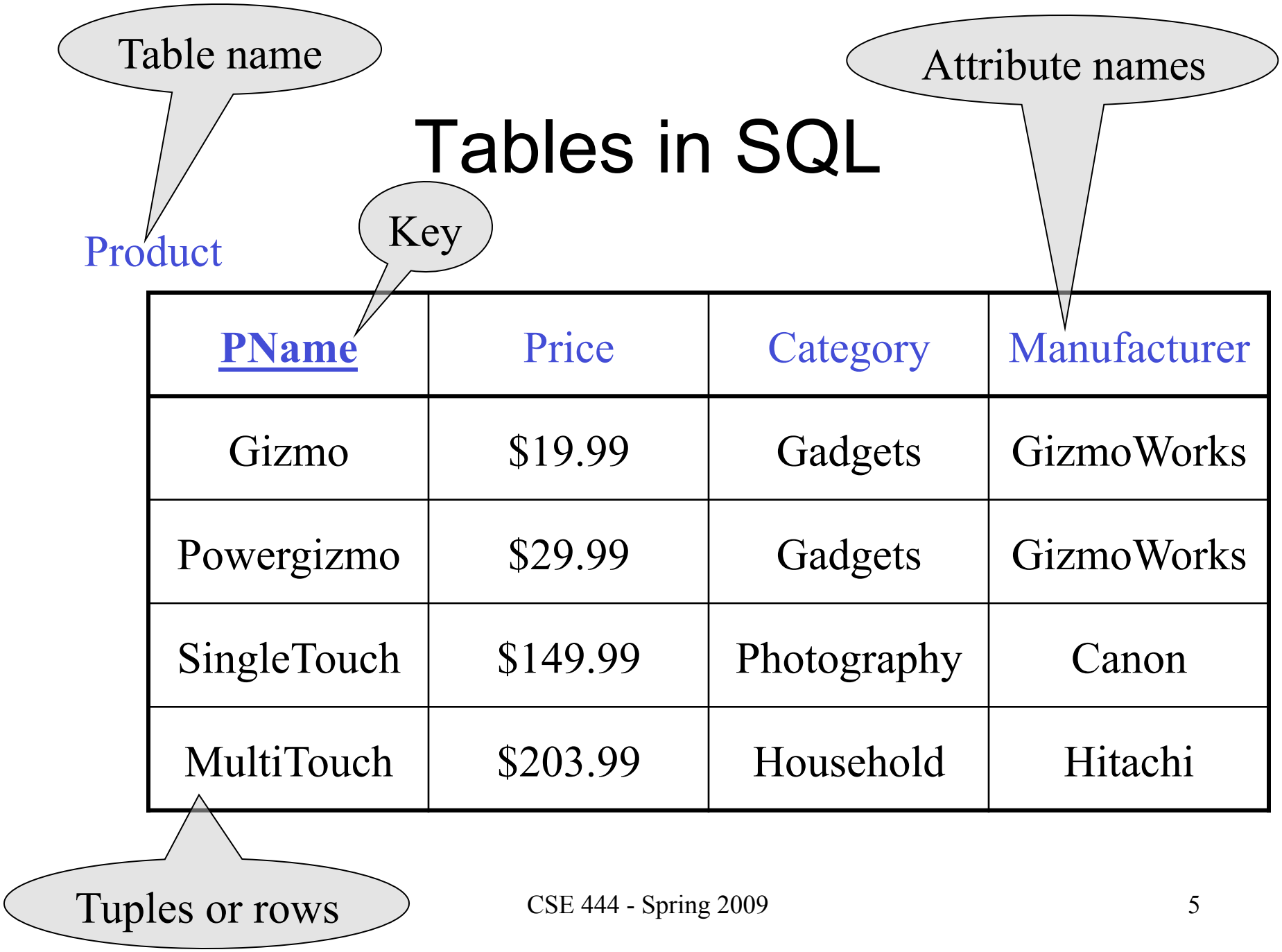
- Data in SQL
- Simple Queries in SQL (6.1)
- Queries with more than one relation (6.2)
- Subqueries (6.3)

Structured Query Language (SQL)

- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
  - Following lectures...

- Data Manipulation Language (DML)
  - Query one or more tables – discussed next !
  - Insert/delete/modify tuples in tables

Tables in SQL



Data Types in SQL

- Atomic types
  - Character strings: CHAR(20), VARCHAR(50)
    - Can be of fixed or variable length
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, …
- Record (aka tuple)
  - Has atomic attributes
- Table (relation)
  - A set of tuples

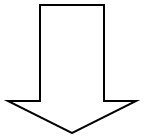
Book Sec. 2.3.2

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

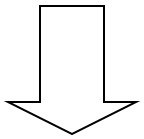
"selection"

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```



"selection" and "projection"

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

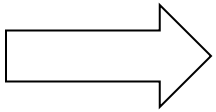
Details

- SQL is case insensitive
  - SELECT = Select = select
  - Product = product
  - BUT ‘Seattle’ ≠ ‘seattle’ (in general)

- Constants must use single quotes
  - ‘abc’ - yes
  - “abc” - no

Eliminating Duplicates

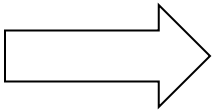
```
SELECT  DISTINCT category
FROM    Product
```



Category
Gadgets
Photography
Household

Compare to:

```
SELECT  category
FROM    Product
```



Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

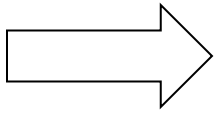
```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='gadgets' AND price > 10
ORDER BY  price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

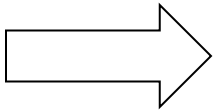
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```



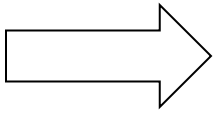
?

```
SELECT   Category
FROM     Product
ORDER BY  PName
```



?

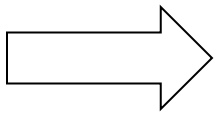
```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```



?

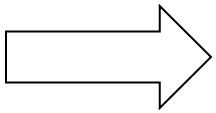
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  DISTINCT category
FROM    Product
ORDER BY category
```



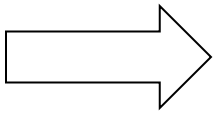
Category
Gadgets
Household
Photography

```
SELECT  Category
FROM    Product
ORDER BY  PName
```



Category
Gadgets
Household
Gadgets
Photography

```
SELECT  DISTINCT category
FROM    Product
ORDER BY PName
```



Error

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

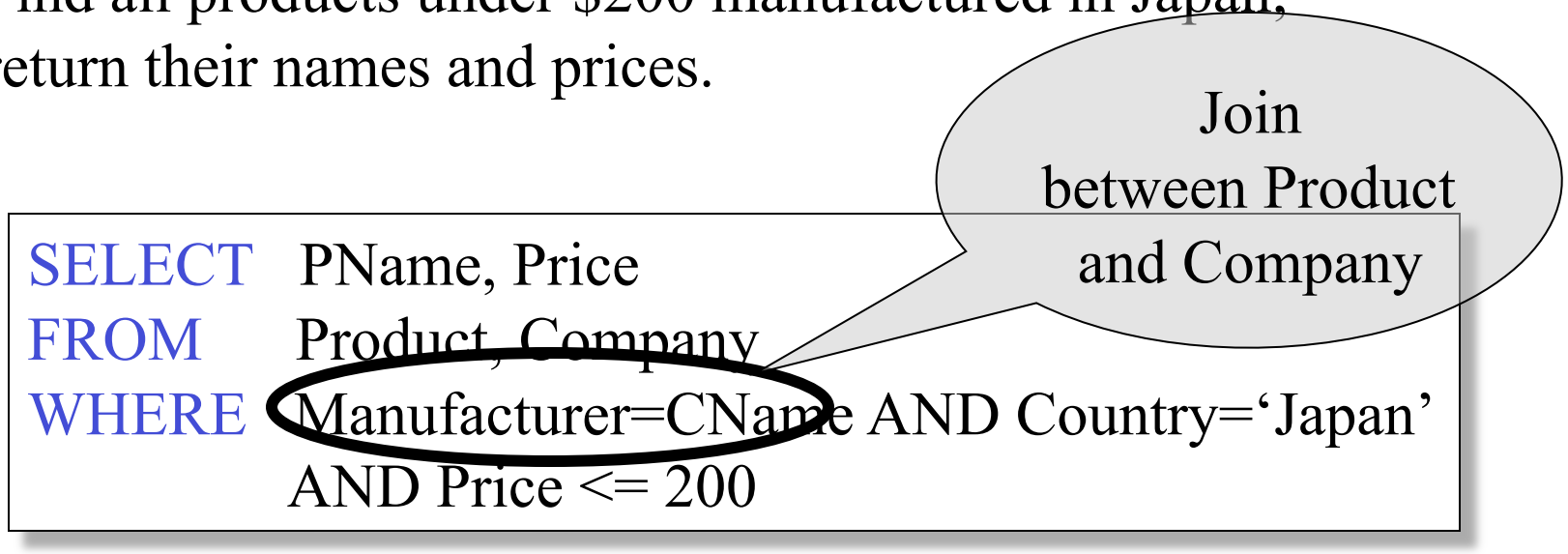
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Foreign key

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.



Joins

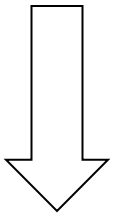
Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```



PName	Price
SingleTouch	$149.99

In Class

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products in the 'toy' category

```
SELECT   cname

FROM

WHERE
```

In Class

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products in the 'toy' category

```
SELECT   DISTINCT cname

FROM Product, Company

WHERE  country = 'China' AND category = 'toy' AND
manufacturer = cname
```

In Class

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products both in the 'electronic' and 'toy' categories

```
SELECT   cname

FROM

WHERE
```

In Class

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products both in the 'electronic' and 'toy' categories

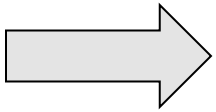
```
SELECT   DISTINCT cname

FROM Product P1, Product P2, Company

WHERE  country = 'China' AND P1.category = 'toy'
AND P2.category = 'electronic' AND P1.manufacturer =
cname  AND P2.manufacturer = cname
```

Tuple Variables

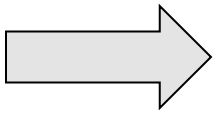
Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?



```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor = Company.cname
```



```
SELECT   DISTINCT x.pname, y.address
FROM     Person AS x, Company AS y
WHERE    x.worksfor = y.cname
```

Meaning (Semantics) of SQL Queries

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer ∪ $\{(a_1,\ldots,a_k)\}$
return Answer

Using the Formal Semantics

What do these queries compute ?

```
SELECT  DISTINCT R.A
FROM    R, S
WHERE   R.A=S.A
```

Returns $R \cap S$

```
SELECT  DISTINCT R.A
FROM    R, S, T
WHERE   R.A=S.A    OR    R.A=T.A
```

Returns $R \cap (S \cup T)$
if $S \neq \phi$ and $T \neq \phi$

Joins Introduce Duplicates

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture some product in the 'Gadgets' category.

```
SELECT   Country
FROM     Product, Company
WHERE    Manufacturer=CName AND Category='Gadgets'
```

Joins Introduce Duplicates

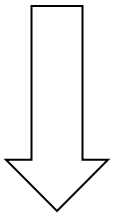
Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  Country
FROM    Product, Company
WHERE   Manufacturer=CName AND Category='Gadgets'
```



Country
USA
USA

Duplicates ! Remember to add DISTINCT

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it's impossible

1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cid=X.cid)
FROM  Product X
```

What happens if the subquery returns more than one city ?

We get a runtime error

1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, city)

Whenever possible, don't use a nested queries:

```
SELECT pname, (SELECT city FROM Company WHERE Company.cid=Product.cid)
FROM  Product
```

=

```
SELECT pname, city
FROM   Product, Company
WHERE Product.cid=Company.cid
```

We have "unnested" the query

1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, city)

Compute the number of products made in by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

Better: we can unnest by using a GROUP BY (next lecture)

2. Subqueries in FROM

Product (pname, price, cid)
Company(cid, cname, city)

Find all products whose prices is > 20 and < 30

```
SELECT P.name
FROM (SELECT * FROM Product WHERE price > 20) as P
WHERE P.price < 30
```

Unnest this query !

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Existential quantifiers

Find all companies that make some products with price < 100

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  EXISTS (SELECT *
                      FROM Product P
                     WHERE C.cid = P.cid and P.price < 100)
```

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Existential quantifiers

Find all companies that make some products with price < 100

Using IN

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE C.cid IN (SELECT P.cid
                FROM Product P
                WHERE P.price < 100)
```

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Existential quantifiers

Find all companies that make some products with price < 100

Using ANY:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE 100 > ANY (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Existential quantifiers

Find all companies that make some products with price < 100

Now let's unnest it:

```
SELECT DISTINCT  C.cname
FROM     Company C, Product P
WHERE  C.cid= P.cid and P.price < 100
```

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Universal quantifiers

Find all companies that make only products with price < 100

same as:

Find all companies whose products all have price < 100

Universal quantifiers are hard ! ☹

3. Subqueries in WHERE

1. Find *the other* companies: i.e. s.t. some product ≥ 100

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price >= 100)
```

2. Find all companies s.t. all their products have price < 100

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid NOT IN (SELECT P.cid
                     FROM Product P
                     WHERE P.price >= 100)
```

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Universal quantifiers

Find all companies that make only products with price < 100

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 100)
```

3. Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Universal quantifiers

Find all companies that make only products with price < 100

Using ALL:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE 100 > ALL  (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```

Question for Database Fans and their Friends

- Can we unnest the *universal quantifier* query ?

Monotone Queries

- A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples

- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics

- Fact: Query with universal quantifier is not monotone

- Consequence: we cannot unnest a query with a universal quantifier

Queries that must be nested

- Queries with universal quantifiers or with negation
- The drinkers-bars-beers example next
- This is a famous example from textbook on databases by Ullman

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL

Find drinkers that frequent some bar that serves some beer they like.

x: $\exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

Find drinkers that frequent only bars that serves some beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

Find drinkers that frequent some bar that serves only beers they like.

x: $\exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Find drinkers that frequent only bars that serves only beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$